arXiv:hep-ph/0005029v1 3 May 2000

Automatic Loop Calculations with *FeynArts*, *FormCalc*, and *LoopTools*

Thomas Hahn[a]

[a]Institut für Theoretische Physik, Universität Karlsruhe
D–76128 Karlsruhe, Germany

KA–TP–7–2000

This article describes three *Mathematica* packages for the automatic calculation of one-loop Feynman diagrams: the diagrams are generated with *FeynArts*, algebraically simplified with *FormCalc*, and finally evaluated numerically using the *LoopTools* package. The calculations are performed analytically as far as possible, with results given in a form well suited for numerical evaluation. The latter is straightforward with the utility programs provided by *FormCalc* (e.g. for translation into Fortran code) and the implementations of the one-loop integrals in *LoopTools*. The programs are also equipped for calculations in supersymmetric models.

1. Introduction

The precision of experimental data achieved at present colliders has in many cases reached or exceeded the per cent level. Obviously a comparable accuracy on the theoretical side is needed in order to draw significant conclusions from such precise measurements. For many observables this means that a one-loop calculation is the lowest acceptable approximation.

Doing one-loop calculations by hand is laborious and error-prone and in some cases simply impossible. So for some time already, software packages have been developed to automate these calculations (e.g. [1, 2]). Incidentally, full automation is possible only up to one loop since no algorithms generic enough for the computation of arbitrary multi-loop Feynman diagrams are known at present. One remaining obstacle is that the existing packages generally tackle only part of the problem, and one still has to spend considerable effort adapting conventions etc. to make them work together.

In this paper the three *Mathematica* packages *FeynArts*, *FormCalc*, and *LoopTools* are presented which work hand in hand. The user has to supply only small driver programs whose main purpose is to specify the necessary input parameters. This makes the whole system very "open" in the sense that the results are returned as *Mathematica* expressions which can easily be manipulated, e.g. to select or modify terms.

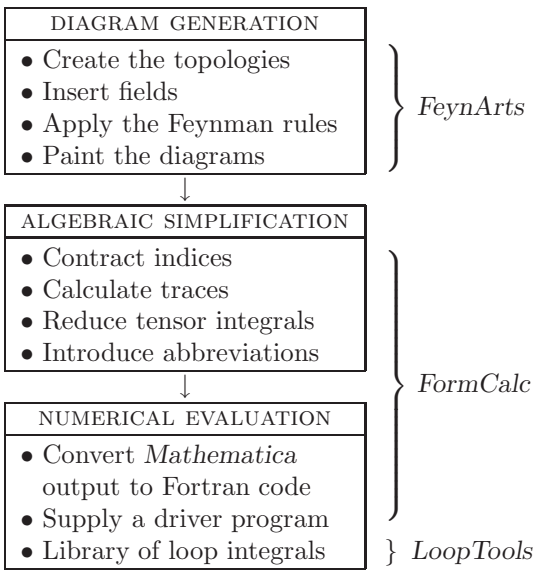

Figure 1. Steps in a one-loop calculation and the distribution of tasks among the programs *FeynArts*, *FormCalc*, and *LoopTools*.

Since one-loop calculations can range anywhere from a handful to several hundreds of diagrams (particularly so in models with many particles like the MSSM), speed is an issue, too. *FormCalc*, the program which does the algebraic simplification, therefore uses *FORM* [5] for the time-consuming parts of the calculation. Owing to *FORM*'s speed, *FormCalc* can process, for example, the 1000-odd one-loop diagrams of W–W scattering in the Standard Model [6] in about 5 minutes on an ordinary Pentium PC.

The following sections describe the main functions of each program. Furthermore, the *FormCalc* package contains two sample calculations in the electroweak Standard Model, $ZZ \to ZZ$ [7] and $e^+e^- \to \bar{t}t$ [8], which demonstrate how the programs are used together.

2. *FeynArts*

FeynArts is a *Mathematica* package for the generation and visualization of Feynman diagrams and amplitudes. The current version 2.2 is a much-expanded version of *FeynArts* 1 [9]. The two most important new features are the generation of counter-term diagrams and the ability to deal with supersymmetric theories (cf. Sect. 5).

FeynArts works in three basic steps, sketched in Fig. 2. The first step is to create all different topologies for a given number of loops and external legs. For example, to create all one-loop topologies for a $1 \to 2$ process, the following call to `CreateTopologies` is used:

```
top = CreateTopologies[1, 1 -> 2]
```

In the second step, the actual particles in the model have to be distributed over the topologies in all allowed ways. E.g. the diagrams for $Z \to b\bar{b}$ are produced with

```
ins = InsertFields[ top,
  V[2] -> {F[4,{3}], -F[4,{3}]} ]
```

where `F[4,{3}]` is the b-quark, `-F[4,{3}]` its antiparticle, and `V[2]` the Z boson.

The fields, their propagators, and their couplings are defined in a special file, the model file, which the user can supply or modify. The following model files are included in *FeynArts*:

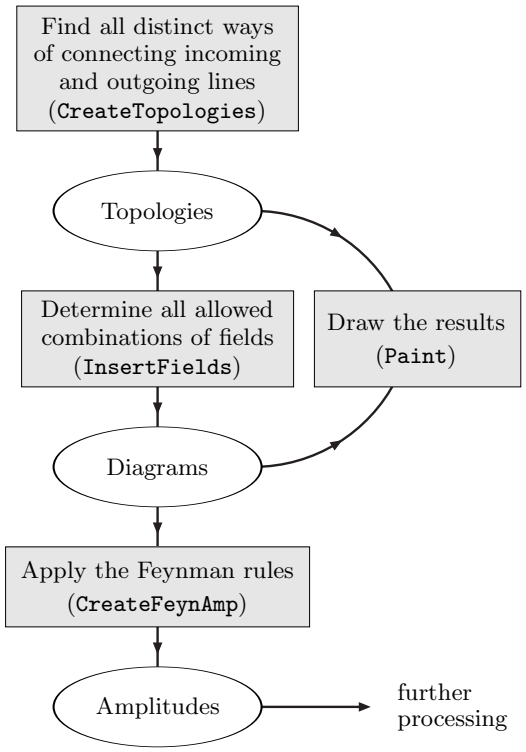

Figure 2. Flowchart for the generation of Feynman amplitudes with *FeynArts*.

the electroweak Standard Model (`SM.mod`) [11], the same including QCD (`SMQCD.mod`), and in the background-field formulation (`SMbgf.mod`). These model files all include the full set of one-loop counter terms. A model file for the MSSM is in preparation.

The diagrams can be drawn with `Paint[ins]`, depending on the options either on screen, or in a PostScript or LaTeX file.

Finally, the analytic expressions for the diagrams are obtained by

```
amp = CreateFeynAmp[ins]
```

An important feature of *FeynArts* is that it distinguishes three levels of fields:

– *Generic level*, e.g. the fermion `F`,
– *Classes level*, e.g. the down-type quark `F[4]`,
– *Particles level*, e.g. the b-quark `F[4,{3}]`.

This is useful for two reasons:

The kinematic structure of a coupling is fixed once the generic fields are specified. For example, all fermion–fermion–scalar couplings are of the form

$$C(F,F,S) = G_-\omega_- + G_+\omega_+$$

where $\omega_\pm = (1 \pm \gamma_5)/2$ are the chirality projectors. This means that most algebraic simplifications like the tensor reduction need to be carried out on the Generic-level amplitude only.

Furthermore, it is more economic to perform index summations (e.g. over the fermion-generation index) in a loop over Classes-level amplitudes instead of generating many Particles-level diagrams.

3. *FormCalc*

FeynArts produces very symbolic output which cannot straightforwardly be implemented in a numerical program. Its evaluation proceeds instead in two steps: first, algebraic simplification in *Mathematica*; then, translation into a Fortran program which computes the squared matrix element and from this the desired quantities (cross-sections, decay rates, asymmetries, etc.).

3.1. Algebraic simplification

The symbolic expressions for the diagrams are simplified algebraically with *FormCalc* which returns the results in a form well suited for numerical evaluation. Specifically, *FormCalc* performs the following simplifications:

- indices are contracted as far as possible,
- fermion traces are evaluated,
- open fermion chains are simplified using the Dirac equation,
- colour structures are simplified using the SU(N) algebra,
- the tensor reduction is done,
- the results are partially factored,
- abbreviations are introduced.

The internal structure of *FormCalc* is simple: it prepares the symbolic expressions of the diagrams in an input file for *FORM*, runs *FORM*, and retrieves the results via the *MathLink* interface (see Fig. 3). This is done completely without intervention by the user, i.e. the user does not see the *FORM* code. *FormCalc* thus combines the speed of *FORM* with the powerful instruction set of *Mathematica* and the latter greatly facilitates further processing of the results.

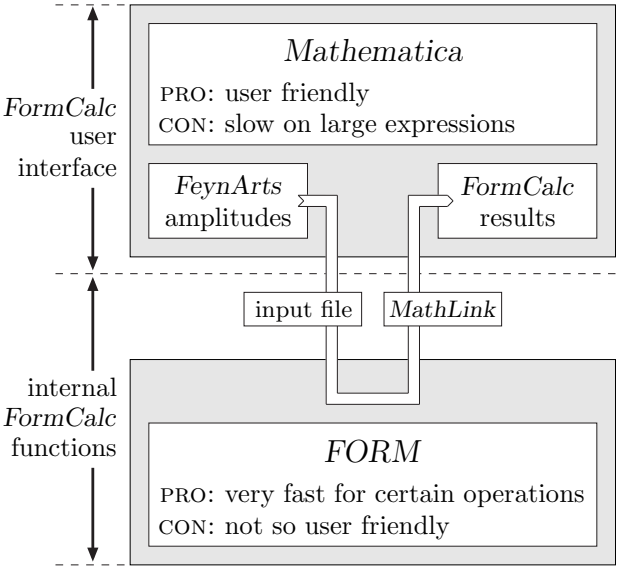

Figure 3. The interplay between *Mathematica* and *FORM* in *FormCalc*.

The main function in *FormCalc* is `OneLoop` (the name is not strictly correct since it works also with tree graphs). It is used like this:

```
<< FormCalc‘
amps = << myamps.m
result = OneLoop[amps]
```

where it is assumed that the file `myamps.m` contains amplitudes generated by *FeynArts*. Note that `OneLoop` needs no declarations of the kinematics of the underlying process; it uses the information *FeynArts* hands down.

Even more comprehensive than `OneLoop`, the function `ProcessFile` can process entire files. It collects the diagrams into blocks such that index summations (e.g. over fermion generations) can later be carried out easily, i.e. only diagrams which are summed over the same indices are put in one block. This nicely complements the genera-

tion of Classes-level diagrams in *FeynArts*, which leaves the index summations to the numerical evaluation in order to reduce the number of diagrams. `ProcessFile` is invoked e.g. as

```
ProcessFile["vertex.amp", "vertex"]
```

which reads the *FeynArts* amplitudes from the input file `vertex.amp` and produces output files of the form `vertex`*id*`.m`, where *id* is some identifier for a particular block.

The output of `OneLoop` or `ProcessFile` is in general a linear combination of loop integrals with prefactors that contain model parameters, kinematic variables, and abbreviations introduced by *FormCalc*. Such abbreviations are introduced for spinor chains, scalar products of vectors, and epsilon tensors contracted with vectors. A term in the output could for instance look like

```
C0i[cc0, MW2, MW2, S, MW2, MZ2, MW2] *
  ( -4 Alfa2 CW2 MW2/SW2 S AbbSum16 +
    32 Alfa2 CW2/SW2 S^2 AbbSum28 +
    4 Alfa2 CW2/SW2 S^2 AbbSum30 -
    8 Alfa2 CW2/SW2 S^2 AbbSum7 +
    Alfa2 CW2/SW2 S (T - U) Abb1 +
    8 Alfa2 CW2/SW2 S (T - U) AbbSum29 )
```

The first line represents the one-loop integral $C_0(M_W^2, M_W^2, s, M_W^2, M_Z^2, M_W^2)$, which is multiplied with a linear combination of abbreviations like `Abb1` or `AbbSum29` with certain coefficients. These coefficients contain kinematical variables like the Mandelstam variables `S`, `T`, and `U` and model parameters, e.g. `Alfa2` $= \alpha^2$.

The automatic introduction of abbreviations is a very important feature which can drastically reduce the size of an amplitude, particularly so because the abbreviations are nested in three levels. Here is an example:

```
AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3

Abb22 = Pair1 Pair3 Pair6

Pair3 = Pair[e[3], k[1]]
```

The definitions of the abbreviations can be retrieved by `Abbreviations[]` which returns a list of rules such that `result //. Abbreviations[]` gives the full, unabbreviated expression.

3.2. Translation to Fortran code

For numerical evaluation, the *Mathematica* expressions produced by *FormCalc* need to be translated into a Fortran program. (They could, in principle, be evaluated in *Mathematica* directly, but this becomes rather slow for large amplitudes.) The translation is done by the program *NumPrep*, which is part of the *FormCalc* package. The philosophy of *NumPrep* is that the user should not have to modify the generated code. This means that the code has to be encapsulated (i.e. no loose ends the user has to bother with), and that all necessary subsidiary files (include files, makefile) have to be produced, too.

From the point of view of the Fortran programmer who wants to use the generated code in his program, the output of *NumPrep* is a single subroutine called

`squared_me`$(k_1, \ldots, k_N, \varepsilon_1, \ldots, \varepsilon_N, \lambda_1, \ldots, \lambda_N)$

which takes as input the external momenta, polarization vectors, and helicities, and returns the squared matrix element. To obtain actual numerical results from the generated code, one needs in addition a driver program whose task is to initialize the model parameters, set up the kinematics, invoke the `squared_me` subroutine, perform necessary phase-space integrations, and finally write the results to a file. A sample driver program for $2 \to 2$ processes is included in *FormCalc*.

Finally, the generated code has to be linked with the *LoopTools* library which provides the one-loop functions.

4. *LoopTools*

LoopTools supplies the actual numerical implementations of the one-loop integrals needed for programs made from the *FormCalc* output. It is based on the reliable package *FF* [10] and provides in addition to the scalar integrals of *FF* also the tensor coefficients in the conventions of [11]. *LoopTools* offers three interfaces: Fortran, C++, and *Mathematica*, so most programming tastes should be served.

Using the *LoopTools* functions in Fortran and C++ is very similar. In Fortran it is necessary to include the file `looptools.h` in every func-

tion or subroutine (for the common blocks). In C++, `clooptools.h` must be included once. Before using any *LoopTools* function, `ffini` must be called and at the end of the calculation `ffexi` may be called to obtain a summary of errors. It is of course possible to change parameters like the scale μ from dimensional regularization; this is described in detail in the manual [12].

A very simple Fortran program would for instance be

```
      program simple
#include "looptools.h"
      call ffini
      print *, B0(1000D0,50D0,80D0)
      call ffexi
      end
```

The C++-version of this program is

```
#include "clooptools.h"

main()
{
  ffini();
  cout << B0(1000.,50.,80.) << "\n";
  ffexi();
}
```

The *Mathematica* interface is even simpler to use:

```
In[1]:= Install["LoopTools"]

In[2]:= B0[1000, 50, 80]

Out[2]= -4.40593 + 2.70414 I
```

5. Calculations in Supersymmetric Models

Special emphasis has been placed on the possibility to do calculations in supersymmetric models with *FeynArts* and *FormCalc*. In particular the following two fundamental problems become relevant in supersymmetric theories:

Problem 1: SUSY theories in general contain Majorana fermions and hence fermion-number-violating couplings (e.g. quark–squark–gluino). The textbook prescription of ordering the Dirac matrices opposite to their occurrence along the arrows on fermionic lines obviously breaks down in this case since one cannot define a fermion-number flow. (Put differently, Majorana-fermion lines have no arrow.)

Solution: *FeynArts* uses the "flipping-rule" algorithm [13]: instead of traversing the fermion lines along the fermion-number flow imposed from the outside, it *chooses* a particular direction for each fermion chain. If it later turns out that, for a Dirac fermion, the chosen direction is opposite to the actual fermion flow, it applies a so-called flipping rule.

Problem 2: Dimensional regularization, the default regularization scheme employed by *FormCalc*, is known to break supersymmetry [14].

Solution: *FormCalc* has two regularization schemes built in which are chosen with the variable `$Dimension`. The default is `$Dimension = D` which corresponds to dimensional regularization. Putting `$Dimension = 4` switches to constrained differential renormalization [3]. The latter technique is equivalent at the one-loop level to regularization by dimensional reduction [4] and is hence suited for calculations in SUSY models.

6. Requirements and Availability

All three packages require *Mathematica* 2.2 or above; *FormCalc* needs in addition *FORM*, preferably version 2 or above; *LoopTools* needs a Fortran compiler and `gcc`/`g++`.

The packages should compile and run without change on any Unix platform. They are specifically known to work under DEC Unix, HP-UX, Linux, Solaris, and AIX. A comprehensive manual which explains installation and usage is included in each package. All three packages are open-source programs and stand under the GNU library general public license. They are available from

```
http://www.feynarts.de
http://www.feynarts.de/formcalc
http://www.feynarts.de/looptools
```

REFERENCES

1. R. Mertig, M. Böhm, and A. Denner, *Comp. Phys. Commun.* **64** (1991) 345.

2. L. Brücher, J. Franzkowski, and D. Kreimer, *Comp. Phys. Commun.* **85** (1995) 153.
3. F. del Aguila, A. Culatti, R. Muñoz Tapia, and M. Pérez-Victoria, *Nucl. Phys.* **B537** (1999) 561.
4. T. Hahn and M. Pérez-Victoria, *Comp. Phys. Commun.* **118** (1999) 153.
5. J.A.M. Vermaseren, *Symbolic Manipulation with FORM*, CAN, Amsterdam (1991).
6. A. Denner and T. Hahn, *Nucl. Phys.* **B525** (1998) 27.
7. A. Denner, S. Dittmaier, and T. Hahn, *Phys. Rev.* **D56** (1997) 117.
8. W. Beenakker, S.C. van der Marck, and W. Hollik, *Nucl. Phys.* **B386** (1991) 24.
9. J. Küblbeck, M. Böhm, and A. Denner, *Comp. Phys. Commun.* **60** (1991) 165;
   T. Hahn, *FeynArts 2.2 user's guide*, available at `http://www.feynarts.de`.
10. G.J. van Oldenborgh and J.A.M. Vermaseren, *Z. Phys.* **C46** (1990) 425.
11. A. Denner, *Fortschr. Phys.* **41** (1993) 307.
12. T. Hahn, *LoopTools user's guide*, available at `http://www.feynarts.de/looptools`.
13. A. Denner, H. Eck, O. Hahn, and J. Küblbeck, *Nucl. Phys.* **B387** (1992) 467.
14. D.M. Capper, D.R.T. Jones, and P. van Nieuwenhuizen, *Nucl. Phys.* **B167** (1980) 479.